JAVA 中正则表达式使用介绍

一、什么是正则表达式

正则表达式是一种可以用于模式匹配和替换的强有力的工具。我们可以在几乎所有的基于 UNIX 系统的工具中找到正则表达式的身影，例如，vi 编辑器，Perl 或 PHP 脚本语言，以及 awk 或 sed shell 程序等。此外，象 JavaScript 这种客户端的脚本语言也提供了对正则表达式的支持。

正则表达式可以让用户通过使用一系列的特殊字符构建匹配模式，进行信息的验证。

此外，它还能够高效地创建、比较和修改字符串，以及迅速地分析大量文本和数据以搜索、移除和替换文本。

例如：

二、基础知识

1.1 开始、结束符号（它们同时也属于定位符）

我们先从简单的开始。假设你要写一个正则表达式规则，你会用到 ^ 和 $ 符号，他们分别是行首符、行尾符。

例如：/^\d+[0-9]?\d+$/

1.2 句点符号

假设你在玩英文拼字游戏，想要找出三个字母的单词，而且这些单词必须以“t”字母开头，以“n”字母结束。另外，假设有一本英文字典，你可以用正则表达式搜索它的全部内容。要构造出这个正则表达式，你可以使用一个通配符——句点符号“.”。这样，完整的表达式就是“t.n”，它匹配“tan”、“ten”、“tin”和“ton”，还匹配“t#n”、“tpn”甚至“t n”，还有其他许多无意义的组合。这是因为句点符号匹配所有字符，包括空格、Tab 字符甚至换行符：

1.3 方括号符号

为了解决句点符号匹配范围过于广泛这一问题，你可以在方括号（“[]”）里面指定看来有意义的字符。此时，只有方括号里面指定的字符才参与匹配。也就是说，正则表达式“t[aeio]n”只匹配“tan”、“Ten”、“tin”和“ton”。但“Toon”不匹配，因为在方括号之内你只能匹配单个字符：

1.4 “或”符号

如果除了上面匹配的所有单词之外，你还想要匹配“toon”，那么，你可以使用“|”操作符。“|”操作符的基本意义就是“或”运算。要匹配“toon”，使用“t(a|e|i|o|oo)n”正则表达式。这里不能使用方扩号，因为方括号只允许匹配单个字符；这里必须使用圆括号“()”。

1.5 表示匹配次数的符号

表一：显示了表示匹配次数的符号，这些符号用来确定紧靠该符号左边的符号出现的次数：

代码/语法	说明
*	重复零次或更多次
+	重复一次或更多次
?	重复零次或一次
{n}	重复 n 次
{n,}	重复 n 次或更多次
{n,m}	重复 n 到 m 次

表二：常用符号

代码/语法	相当于
\w	[0-9A-Za-z_]
\W	[^0-9A-Za-z_]
\s	[\t\n\r\f]
\S	[^\t\n\r\f]
\d	[0-9]
\D	[^0-9]

表二中的符号意义：

- \w

包括下划线的字母和数字。等同于[0-9A-Za-z_]。

若为匹配多字节字符的正则表达式时，则也会匹配日语的全角字符。

- \W

非字母和数字。\w 以外的单个字符。

- \s

空字符。相当于[\t\n\r\f]

- \S

非空字符。[\t\n\r\f] 以外的单个字符。

- \d

数字。即[0-9]

- \D

非数字。\d 以外的单个字符

1.6 定位符介绍（用于规定匹配模式在目标对象中的出现位置）

较为常用的定位符包括： “ˆ”，“$”，“\b” 以及 “\B”。其中，“ˆ”定位符规定匹配模式必须出现在目标字符串的开头，“$”定位符规定匹配模式必须出现在目标对象的结尾，\b 定位符规定匹配模式必须出现在目标字符串的开头或结尾的两个边界之一，而“\B”定位符则规定匹配对象必须位于目标字符串的开头和结尾两个边界之内，即匹配对象既不能作为目标字符串的开头，也不能作为目标字符串的结尾。同样，我们也可以把“ˆ”和“$”以及“\b”和“\B”看作是互为逆运算的两组定位符。举例来说：

/ˆhell/

因为上述正则表达式中包含“ˆ”定位符，所以可以与目标对象中以“hell”，“hello”或 “hellhound”开头的字符串相匹配。

/ar$/

因为上述正则表达式中包含“$”定位符，所以可以与目标对象中以“car”，“bar”或 “ar” 结尾的字符串相匹配。

/\bbom/

因为上述正则表达式模式以“\b”定位符开头，所以可以与目标对象中以“bomb”，或 “bom”开头的字符串相匹配。

/man\b/

因为上述正则表达式模式以“\b”定位符结尾，所以可以与目标对象中以“human”，“woman”或 “man”结尾的字符串相匹配。

为了能够方便用户更加灵活的设定匹配模式，正则表达式允许使用者在匹配模式中指定某一个范围而不局限于具体的字符。例如：

/[A-Z]/

上述正则表达式将会与从 A 到 Z 范围内任何一个大写字母相匹配。

/[a-z]/

上述正则表达式将会与从 a 到 z 范围内任何一个小写字母相匹配。

/[0-9]/

上述正则表达式将会与从 0 到 9 范围内任何一个数字相匹配。

/([a-z][A-Z][0-9])+/

上述正则表达式将会与任何由字母和数字组成的字符串，如 “aB0” 等相匹配。这里需要提醒用户注意的一点就是可以在正则表达式中使用 “()” 把字符串组合在一起。“()”符号包含的内容必须同时出现在目标对象中。因此，上

述正则表达式将无法与诸如 “abc”等的字符串匹配，因为“abc”中的最后一个字符为字母而非数字。

如果我们希望在正则表达式中实现类似编程逻辑中的“或”运算，在多个不同的模式中任选一个进行匹配的话，可以使用管道符 “|”。例如：

/to|too|2/

上述正则表达式将会与目标对象中的 “to”，“too”，或 “2” 相匹配。

正则表达式中还有一个较为常用的运算符，即否定符 “[^]”。与我们前文所介绍的定位符 “^” 不同，否定符 “[^]”规定目标对象中不能存在模式中所规定的字符串。例如：

/[^A-C]/

上述字符串将会与目标对象中除 A，B，和 C 之外的任何字符相匹配。一般来说，当“^”出现在 “[]”内时就被视做否定运算符；而当“^”位于“[]”之外，或没有“[]”时，则应当被视做定位符。

最后，当用户需要在正则表达式的模式中加入元字符，并查找其匹配对象时，可以使用转义符“\”。例如：

/Th*/

上述正则表达式将会与目标对象中的“Th*”而非“The”等相匹配。

三、正则表达式规则的例子

```
/^(\d{3}-|\d{4}-)?(\d{8}|\d{7})?$/     //国内电话
/^[1-9]*[1-9][0-9]*$/     //腾讯 QQ
/^[\\w-]+(\\.[\\w-]+)*@[\\w-]+(\\.[\\w-]+)+$/          //email 地址
/^[a-zA-Z]+://(\\w+(-\\w+)*)(\\.(\\w+(-\\w+)*))*(\\?\\s*)?$/     //url
/^\d+$/      //非负整数
/^[0-9]*[1-9][0-9]*$/     //正整数
/^((-\\d+)|(0+))$/     //非正整数
/^-[0-9]*[1-9][0-9]*$/     //负整数
/^-?\\d+$/           //整数
/^\\d+(\\.\\d+)?$/     //非负浮点数
/^(([0-9]+\\.[0-9]*[1-9][0-9]*)|([0-9]*[1-9][0-9]*\\.[0-9]+)|([0-9]*[
1-9][0-9]*))$/     //正浮点数
/^((-\\d+(\\.\\d+)?)|(0+(\\.0+)?))$/     //非正浮点数
/^(-(([0-9]+\\.[0-9]*[1-9][0-9]*)|([0-9]*[1-9][0-9]*\\.[0-9]+)|([0-9]
*[1-9][0-9]*)))$/  //负浮点数
/^(-?\\d+)(\\.\\d+)?$/     //浮点数
/^[a-zA-Z]+$/     //由 26 个英文字母组成的字符串
/^[a-z]+$/     //由 26 个英文字母的大写组成的字符串
/^[a-z]+$/     //由 26 个英文字母的小写组成的字符串
/^[a-za-z0-9]+$/     //由数字和 26 个英文字母组成的字符串
/^\\w+$/     //由数字、26 个英文字母或者下划线组成的字符串
```

/^\d+[.]?\d+$/　　//可以有小数点的任意多数字（全部为数字）

/(?=^[0-9a-zA-Z]{4,20}$)\w*[a-zA-Z]+\w*/　　//同时满足下面三个条件

（1）　　数字和字母（2）4-20 位（3）不能全部是数字

四、应用

1. 应用于 JavaScript （用来验证）

```
function doCheck(){
var patrn = /^\d+[.]?\d+$/;
var vf1 = document.queryForm.f2Text.value; //文本框
var vf2 = document.queryForm.f4Text.value;
var vf3 = document.queryForm.f6Text.value;
var va1 = document.queryForm.a2.checked;      //单选按钮
var va2 = document.queryForm.a4.checked;
var va3 = document.queryForm.a6.checked;
if(va1){
    if(!patrn.exec(vf1)){
      alert("请您输入数字，如：30 、 5.8");
      return;
    }
}
if(va2){
    if(!patrn.exec(vf2)){
      alert("请您输入数字，如：30 、 5.8");
      return;
```

```
    }
}
。。。 。。。
}
```

2. 在 Java 中的应用

Java 包 java.util.regex 提供对正则表达式的支持。而且 Java.lang.String 类中的 replaceAll 和 split 函数也是调用的正则表达式来实现的。

正则表达式对字符串的操作主要包括：字符串匹配，指定字符串替换，指定字符串查找和字符串分割。下面就用一个例子来说明这些操作是如何实现的：

```
<%@ page import="java.util.regex.*"%>;
<%
Pattern p=null; //正则表达式
Matcher m=null; //操作的字符串
boolean b;
String s=null;
StringBuffer sb=null;
int i=0;
//字符串匹配，这是不符合的
   p = Pattern.compile("a*b");
   m = p.matcher("baaaaab");
   b = m.matches();
out.println(b+"<br>;");
//字符串匹配，这是符合的
```

```
    p = Pattern.compile("a*b");
    m = p.matcher("aaaaab");
    b = m.matches();
    out.println(b+"<br>;");
//字符串替换
    p = Pattern.compile("ab");
    m = p.matcher("aaaaab");
    s = m.replaceAll("d");
    out.println(s+"<br>;");
    p = Pattern.compile("a*b");
    m = p.matcher("aaaaab");
    s = m.replaceAll("d");
    out.println(s+"<br>;");
    p = Pattern.compile("a*b");
    m = p.matcher("caaaaab");
    s = m.replaceAll("d");
    out.println(s+"<br>;");
//字符串查找
p = Pattern.compile("cat");
m = p.matcher("one cat two cats in the yard");
sb = new StringBuffer();
while (m.find()) {
```

```
        m.appendReplacement(sb, "dog");
        i++;
}
m.appendTail(sb);
out.println(sb.toString()+"<br>;");
out.println(i+"<br>;");
i=0;
p = Pattern.compile("cat");
m = p.matcher("one cat two ca tsi nthe yard");
   sb = new StringBuffer();
while (m.find()) {
        m.appendReplacement(sb, "dog");
        i++;
}
m.appendTail(sb);
out.println(sb.toString()+"<br>;");
out.println(i+"<br>;");



p = Pattern.compile("cat");
m = p.matcher("one cat two cats in the yard");
p=m.pattern();
```

```
m = p.matcher("bacatab");

b = m.matches();

out.println(b+"<br>;");

s = m.replaceAll("dog");

out.println(s+"<br>;");


i=0;

p = Pattern.compile("(fds){2,}");

m = p.matcher("dsa da fdsfds aaafdsafds aaf");

   sb = new StringBuffer();

while (m.find()) {

       m.appendReplacement(sb, "dog");

       i++;

}

m.appendTail(sb);

out.println(sb.toString()+"<br>;");

out.println(i+"<br>;");


   p = Pattern.compile("cat");

   m = p.matcher("one cat two cats in the yard");

   sb = new StringBuffer();

   while (m.find()) {
```

```
        m.appendReplacement(sb, "<font color=\"red\">;cat</font>;");
    }
m.appendTail(sb);
out.println(sb.toString()+"<br>;");
String aa=sb.toString();
out.println(aa+"<br>;");
//字符串分割
    p = Pattern.compile("a+");
    String[] a=p.split("caaaaaat");
    for(i=0;i<a.length;i++)
    {
    out.println(a+"<br>;");
    }
    p = Pattern.compile("a+");
    a=p.split("c aa aaaa t",0);
    for(i=0;i<a.length;i++)
    {
    out.println(a+"<br>;");
    }
    p = Pattern.compile(" +");
    a=p.split("c aa      aaaa t",0);
    for(i=0;i<a.length;i++)
```

```
{
out.println(a+"<br>;");
}
p = Pattern.compile("\\+");
a=p.split("dsafasdfdsafsda+dsagfasdfa+sdafds");
out.println(a.length+"<br>;");
for(i=0;i<a.length;i++)
{
out.println(a+"<br>;");
}
```